UMBC

AN HONORS UNIVERSITY IN MARYLAND

IS 450/IS 650–
Data Communications and Networks

Course Review
Midterm Exam

Nirmalya Roy
Department of Information Systems
University of Maryland Baltimore County

Midterm Exam

- When: Tuesday (3/31) 4:30pm - 6:30pm
- Where: In Class

- Closed book, Closed notes
- Computer Networks and the Internet (Chapter 1); Application Layer (Chapter 2) and Transport Layer (up to Chapter 3.3)

- Material for preparation:
  - Lecture Slides
  - Quizzes
  - Textbooks
    - Computer Networking: A Top Down Approach,

Course Overview

- Computer Networks and the Internet (Chapter 1)
  - Packet and circuit switching
  - End to End Delay
- Application Layer (Chapter 2)
  - Web, HTTP, FTP, SMTP, DNS, Peer-peer and Socket
- Transport Layer (Chapter 3)
  - Multiplexing & Demultiplexing

Chapter 1: Computer Networks & Internet

1.1 What *is* the Internet?

1.2 Network edge

1.3 Network access and physical media

1.4 Network core

1.5 Internet structure and ISPs

1.6 Delay and loss in packet-switched networks

1.7 Protocol layers, service models

1.8 History

Chapter 1: Roadmap

1.1 What *is* the Internet?

1.2 Network edge

1.3 Network access and physical media

1.4 Network core

1.5 Internet structure and ISPs

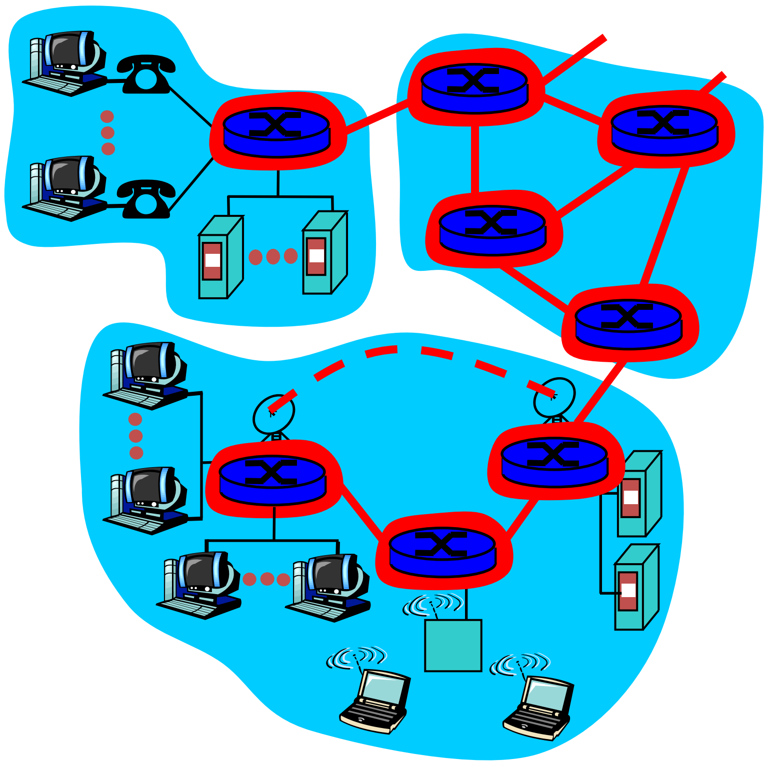
1.6 Delay & loss in packet-switched networks

1.7 Protocol layers, service models

1.8 History

The Network Core

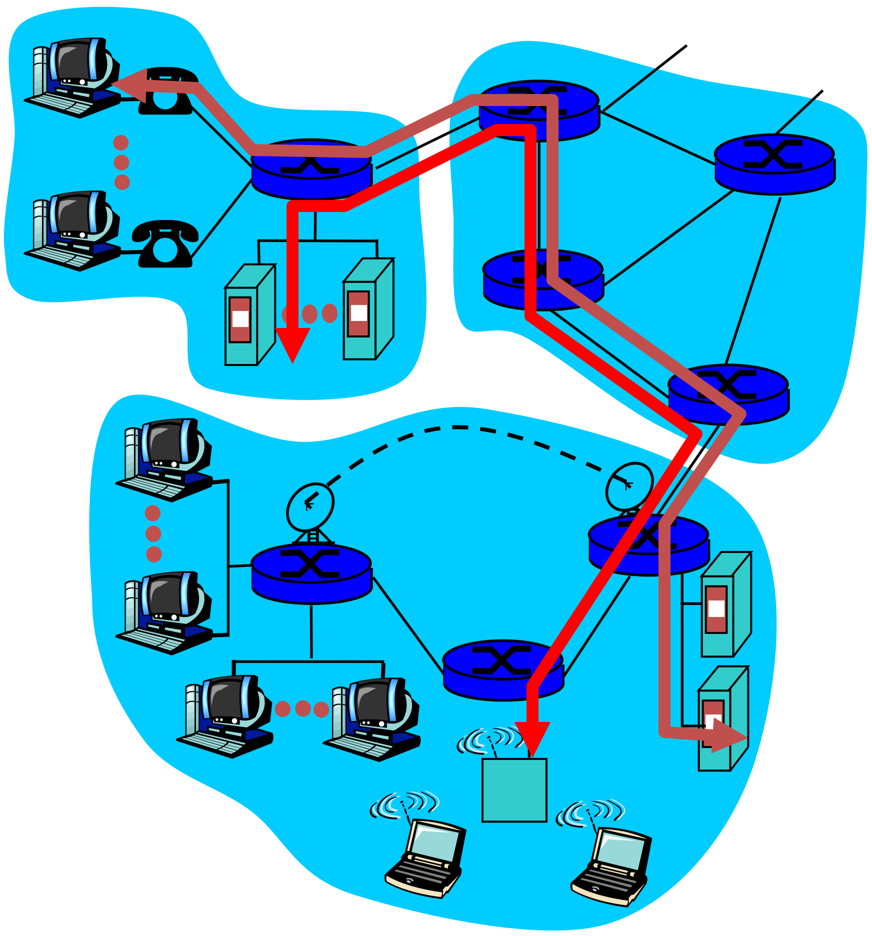
- mesh of interconnected routers
- *the* fundamental question: how is data transferred through net?
  - circuit switching: dedicated circuit per call: telephone net
  - packet-switching: data sent thru net in discrete "chunks"
  - Forwarding table and routing protocols



Network Core: Circuit Switching

End-end resources reserved for "call"

- link bandwidth, switch capacity
- dedicated resources: no sharing
- circuit-like (guaranteed) performance
- call setup required



Network Core: Circuit Switching

Network resources (e.g., bandwidth) divided into "pieces"

- pieces allocated to calls
- resource piece *idle* if not used by owning call *(no sharing)*

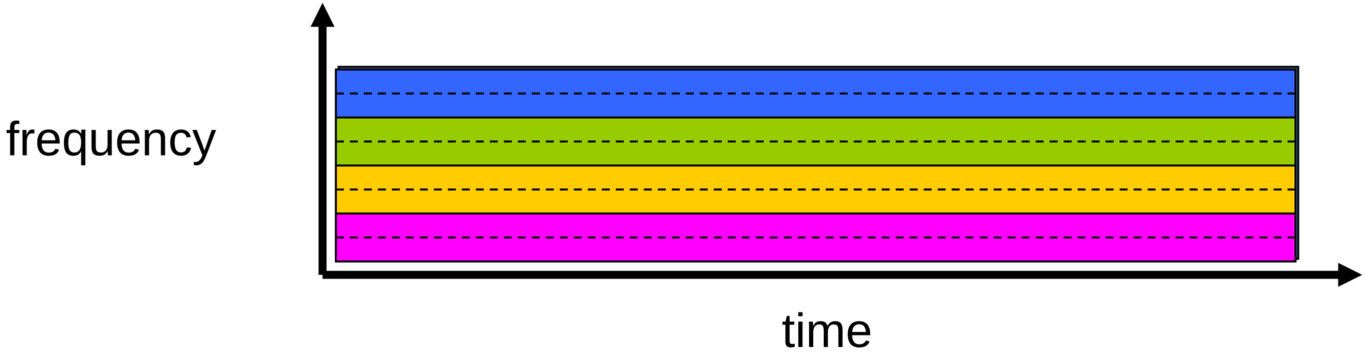
- dividing link bandwidth into "pieces"
  - frequency division
  - time division

Circuit Switching: FDM and TDM

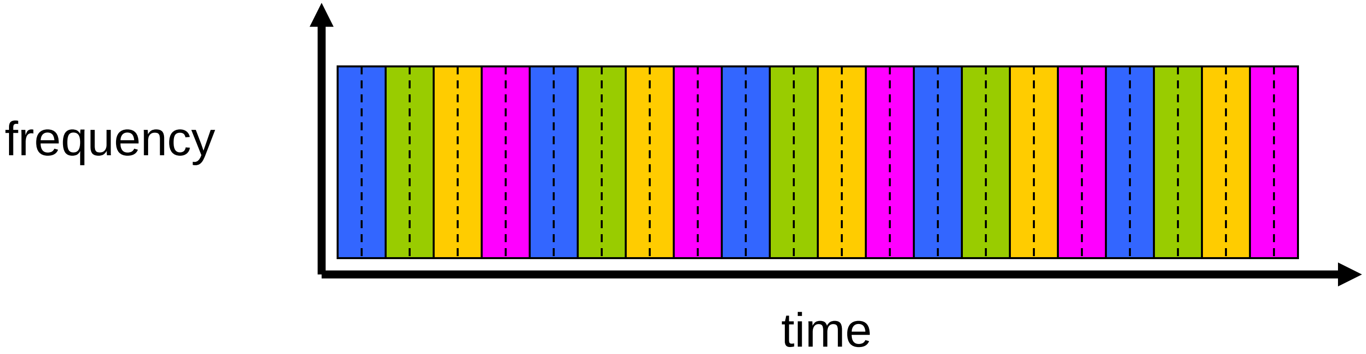
Example:

4 users

FDM



TDM



FDM vs TDM

- What are the tradeoffs?
  - Advantage and disadvantage of dividing frequency ?
  - Advantage and disadvantage of dividing time ?

Numerical example

- How long does it take to send a file of 640,000 bits from host A to host B over a circuit-switched network?
  - All links are 1.536 Mbps
  - Each link uses TDM with 24 slots/sec
  - 500 msec to establish end-to-end circuit

Let's work it out!

Network Core: **Packet Switching**

each end-end data stream divided into *packets*

- user A, B packets *share* network resources
- each packet uses full link bandwidth
- resources used *as needed*

~~Bandwidth division into "pieces"~~

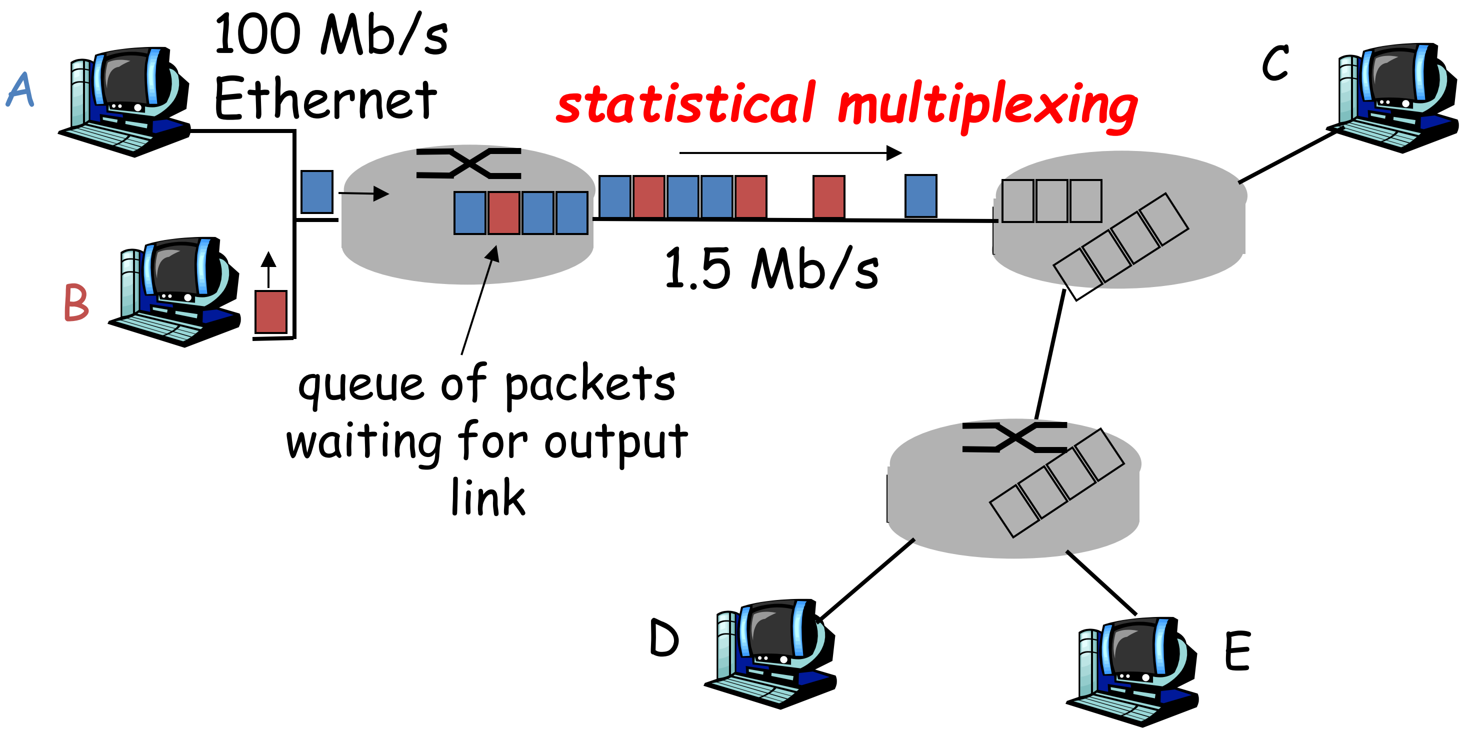
~~Dedicated allocation~~

~~Resource reservation~~

resource contention:

- aggregate resource demand can exceed amount available
  - Packets queue up
- store and forward: packets move one hop at a time
  - Node receives complete packet before forwarding

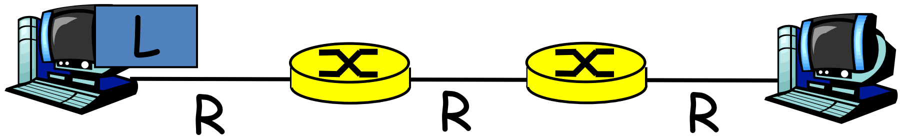
Packet Switching: Statistical Multiplexing



Sequence of A & B packets does not have fixed pattern, shared on demand ***statistical multiplexing***.

TDM: each host gets same slot in revolving TDM frame.

Packet-switching: store-and-forward



- Takes L/R seconds to transmit (push out) packet of L bits on to link of R bps
- Entire packet must arrive at router before it can be transmitted on next link: *store and forward*
- delay = 3L/R (assuming zero propagation delay)

Example:

- L = 7.5 Mbits
- R = 1.5 Mbps
- delay = 15 sec

more on delay shortly …

Packet-switched networks: forwarding

- *Goal:* move packets through routers from source to dest.
  - we'll study several path selection (routing) algorithms (chap 4)

- datagram network:
  - *destination address* in packet determines next hop
  - routes may change during session
  - analogy: driving, asking directions

- virtual circuit network:
  - packet carries tag (virtual circuit ID), tag determines next hop
  - fixed path determined at *call setup time*, remains fixed thru call
  - *routers maintain per-call state*
  - (analogy: air trains in airports)

Compare

Thoughts on tradeoffs between packet switching and circuit switching?

Which one would you take?

Under what circumstances?

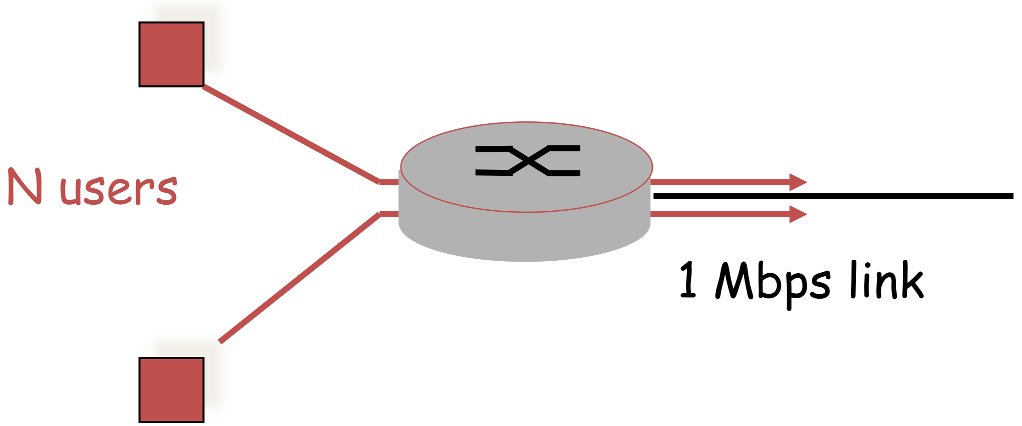
Why?

Packet switching versus Circuit switching

Packet switching allows more users to use network!

- problem: 1 Mbps link
- each user:
  - 100 kbps when "active"
  - active 10% of time
- circuit-switching:
  - 10 users
- packet switching (ps):
  - with 35 users,

probability > 10 active users is less than 0.0004



Q: how did we get value 0.0004?
Get performance of circuit switching with 3 times more users in case of PS

Packet switching versus Circuit switching

Is packet switching a "slam dunk winner?"

- Great for absorbing bursty data from individual sources
  - resource sharing (due to diversity)
  - simpler, no call setup
- Excessive congestion: packet delay and loss
  - protocols needed for reliability, congestion control

Why?

- Q: How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps
  - still unsolved (chapter 7)

Problem on Circuit and Packet switching

- Suppose users share a 15 Mbps link. Also suppose each user requires 1 Mbps when transmitting, but each user transmit only 10% time.

a) When circuit switching is used, how many users can be supported?

b) Suppose there are 30 users. Find the probability that any given time, exactly 20 users are transmitting simultaneously. (Hint: Use the binomial distribution)

- Solve this problem from Quiz 1

Binomial Probability Formula

$$P(r) = {}^{n}C_{r}\, p^{r}(1-p)^{n-r} = \frac{n!}{r!(n-r)!}\, p^{r} q^{n-r}$$

for $r = 0, 1, 2, \ldots, n$

where

n = number of trials

r = number of successes among n trials

p = probability of success in any one trial

q = probability of failure in any one trial ($q = 1 - p$)

Chapter 1: Roadmap

1.1 What *is* the Internet?

1.2 Network edge

1.3 Network access and physical media

1.4 Network core

1.5 Internet structure and ISPs

1.6 Delay & loss in packet-switched networks

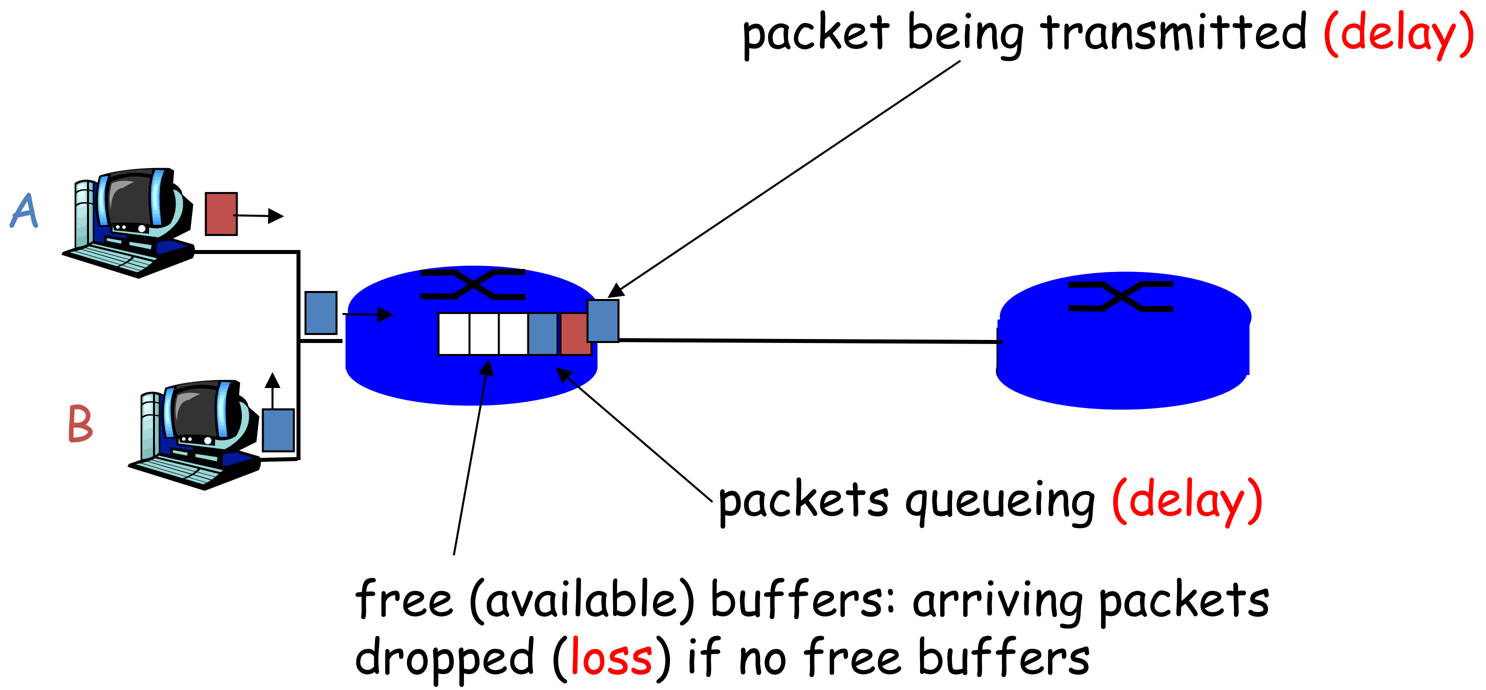
1.7 Protocol layers, service models

1.8 History

How do loss and delay occur?

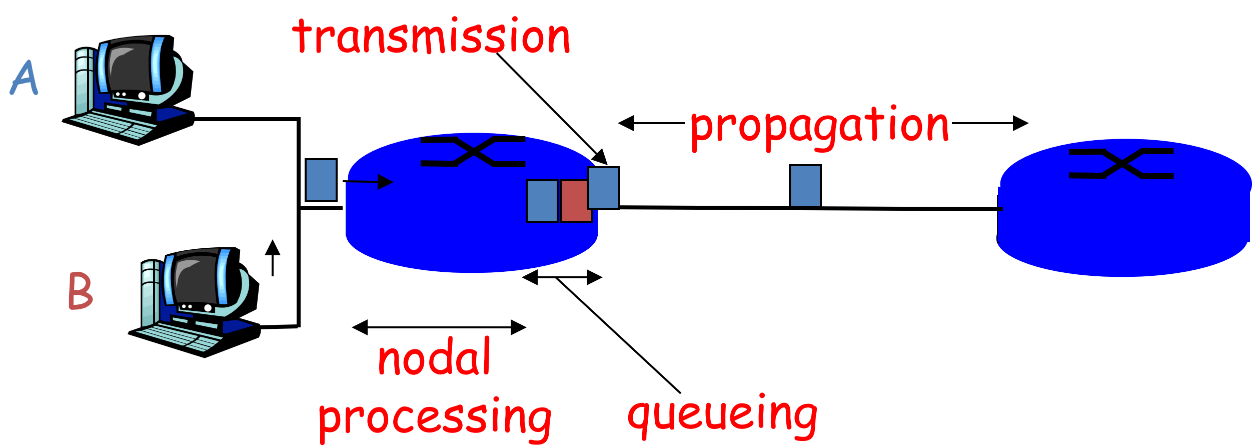
packets *queue* in router buffers

- packet arrival rate to link exceeds output link capacity
- packets queue, wait for turn



Four Sources of Packet Delay

- 1. nodal processing:
  - check bit errors
  - determine output link
- 2. queueing:
  - time waiting at output link for transmission
  - depends on congestion level of router



Delay in packet-switched networks

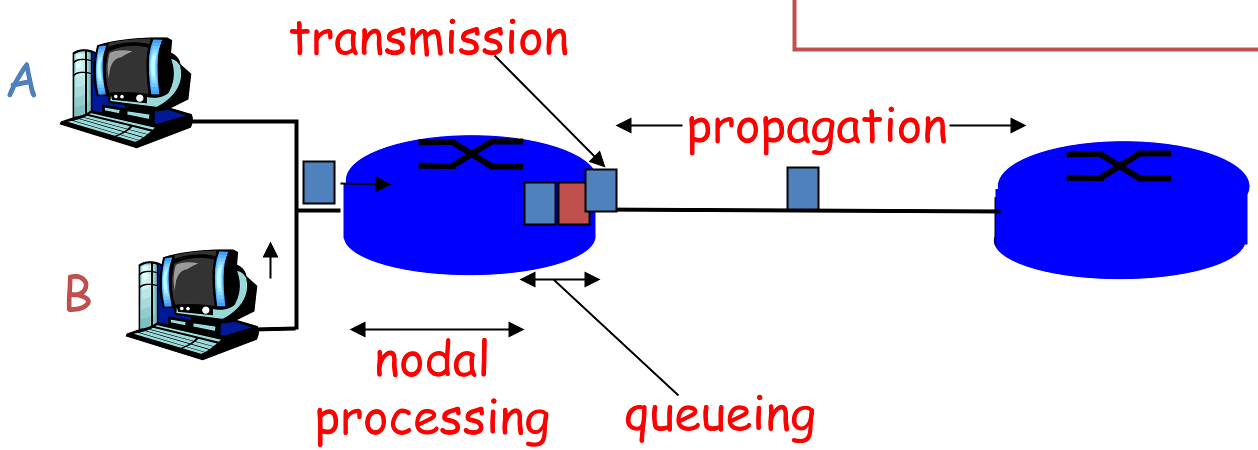
3. Transmission delay:

- L = packet length (bits)
- R = link bandwidth (bps)
- time to send bits into link = L/R

4. Propagation delay:

- d = length of physical link
- s = propagation speed in medium (~2×10^8 m/sec)
- propagation delay = d/s

Note: R and s are *very* different quantities!



Comparing Transmission & Propagation Delays

- Transmission delay
  - Amount of time required to push out a packet
  - Function of the packet’s length & transmission rate of the link
  - Nothing to do with the distance between the two routers
- Propagation delay
  - Time it takes a bit to propagate from one router to the next
  - Function of the distance between two routers and propagation speed
  - Nothing to do with the packets’ length or transmission rate

Nodal delay

$$\boxed{d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}}$$

- d_{proc} = processing delay
  - typically a few microsecs or less
- d_{queue} = queuing delay
  - depends on congestion
- d_{trans} = transmission delay
  - = L/R, significant for low-speed links
- d_{prop} = propagation delay
  - a few microsecs to hundreds of msecs

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

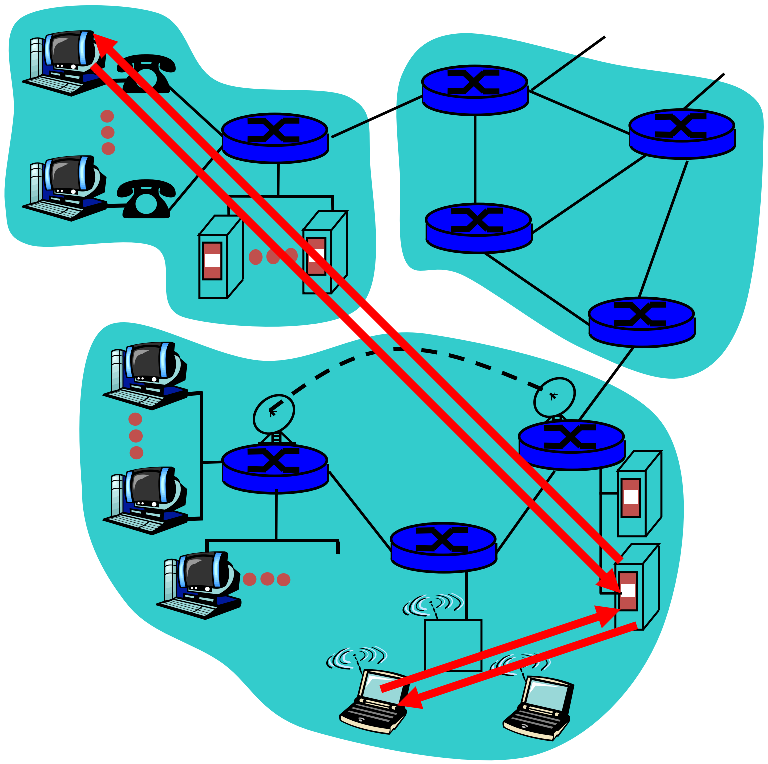
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
    - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture



server:

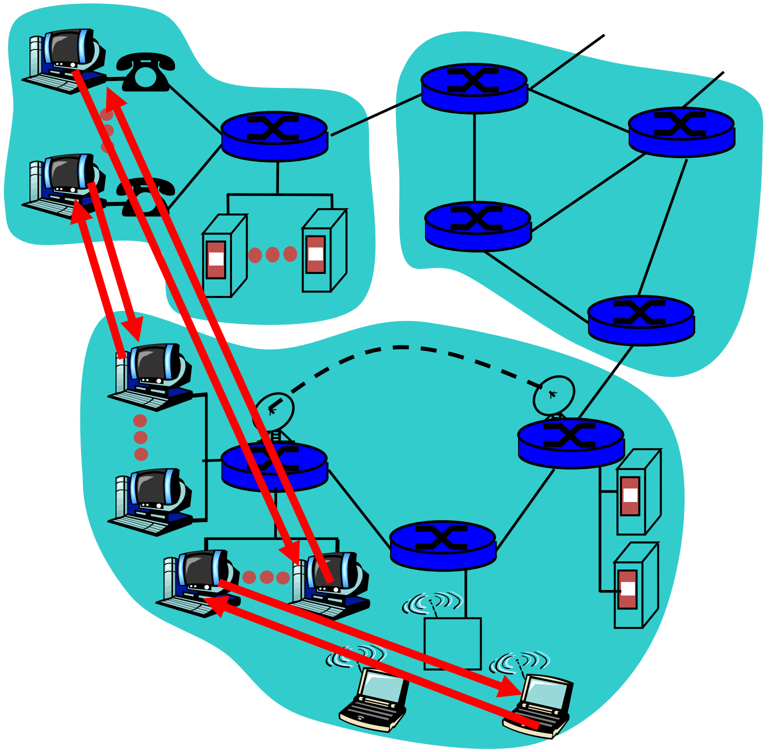
- always-on host
- permanent IP address
- server farms for scaling
- data centers

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- Major Challenges: ISP friendly, Security, Incentives
- example: Gnutella (peer-to-peer file sharing network)



Highly scalable but difficult to manage

Hybrid of client-server and P2P

Skype

- Internet telephony app
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: Why bother? Why is there a UDP?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)	typically UDP

Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

Web and HTTP

First some jargon

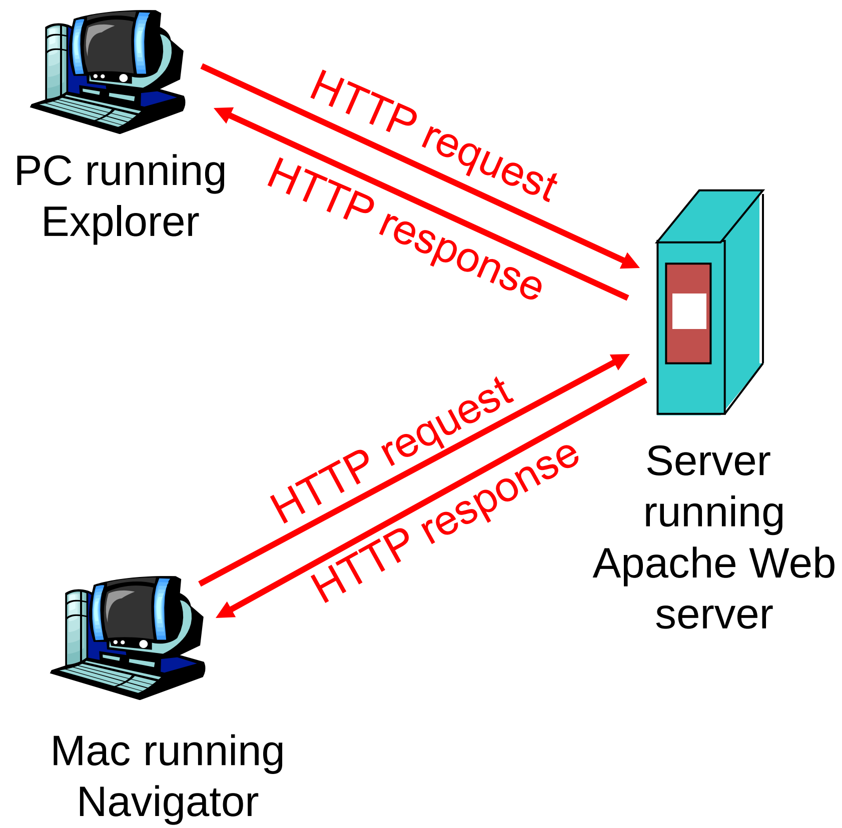
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several **referenced objects**
- Each object is addressable by a URL
- Example URL:

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP overview (continued)

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- ❒ past history (state) must be maintained
- ❒ if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

Nonpersistent HTTP

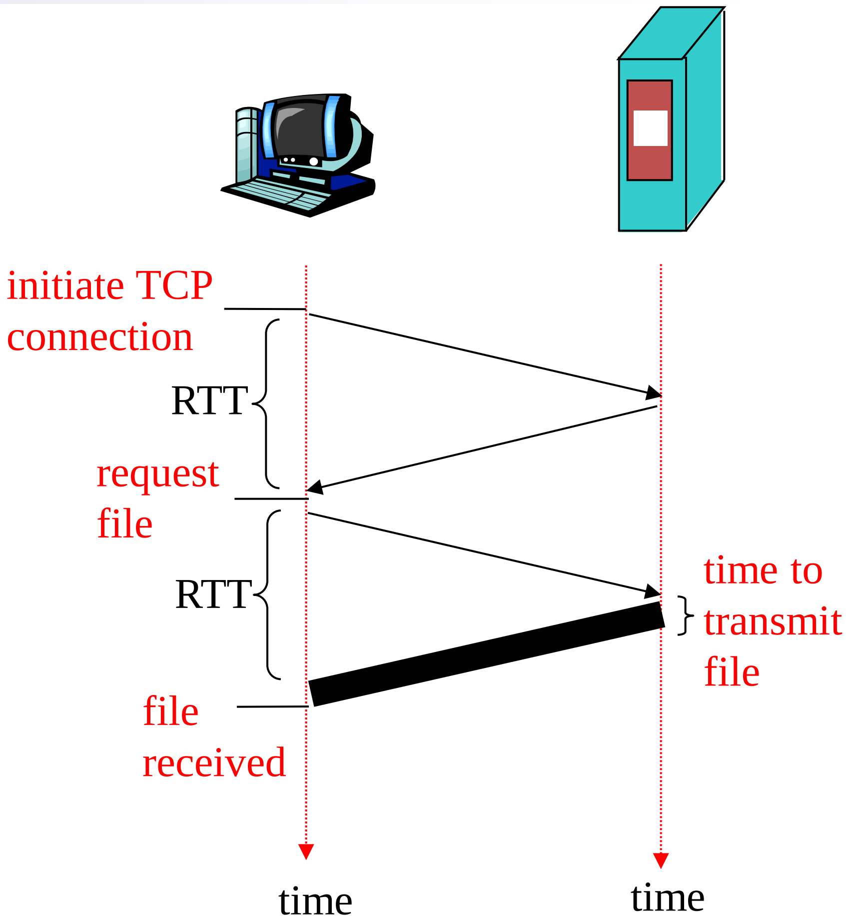
- At most one object is sent over a TCP connection
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server
- HTTP/1.1 uses persistent connections in default mode

Non-Persistent HTTP: Response time

Round Trip Time (RTT) = time to send a small packet to travel from client to server and back.



Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+ <file transmit time>

Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch **referenced objects**

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection

Persistent ***without*** pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent ***with*** pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:

- ***distributed database*** implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS

<u>DNS services</u>

- Hostname to IP address translation
- Host aliasing
  - Canonical and alias names
  - Alias: `enterprise.com` or `www.enterprise.com`

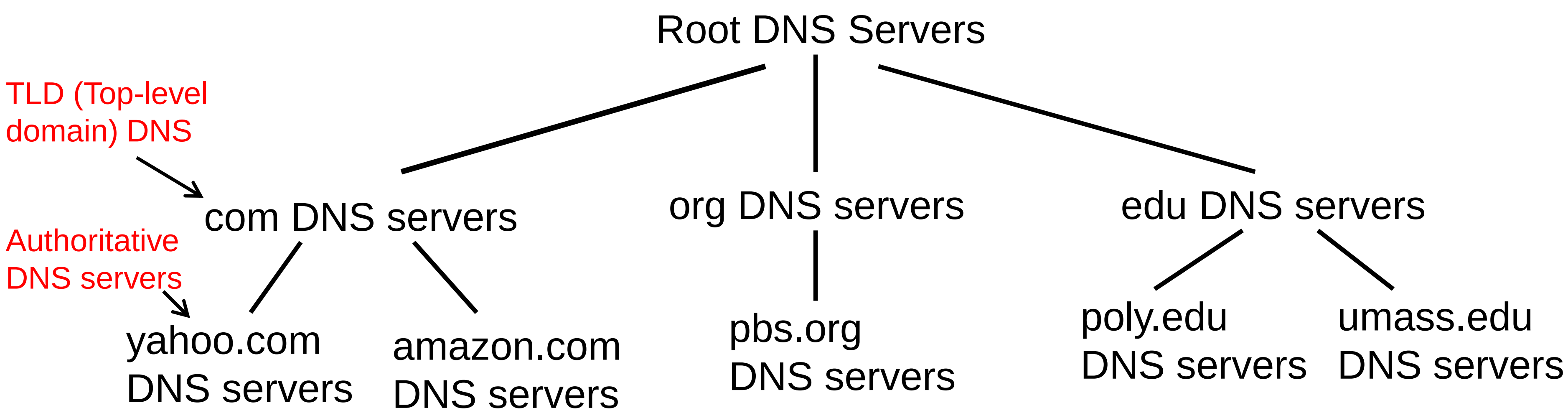
    Canonical:`relay1.west coast.enterprise.com`
- Load distribution
  - Replicated Web servers: set of IP addresses for one canonical name

<u>Why not centralize DNS?</u>

- single point of failure
- traffic volume
- distant centralized database
- Maintenance

doesn't *scale!*

Distributed, Hierarchical Database



Client wants IP for www.amazon.com; 1st approx:

- Client queries a root server to find .com DNS server
- Client queries com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

DNS records

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

- Type=A
  - **`name`** is hostname
  - **`value`** is IP address
- Type=NS
  - **`name`** is domain (e.g. foo.com)
  - **`value`** is hostname of authoritative name server for this domain (e.g. dns.foo.com)
- Type=CNAME
  - **`name`** is alias name for some "canonical" (the real) name
    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **`value`** is canonical name
- Type=MX
  - **`value`** is name of mailserver associated with **`name`** (e.g. foo.com, mail.bar.foo.com, MX)

Inserting records into DNS

- Example: just created startup “Network Utopia”
- Register name networkutopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for mail.networkutopia.com
- How do people get the IP address of your Web site?

Chapter 3 Outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Transport Layer (Chapter 3)

- Transport Layer
- Multiplexing / Demultiplexing

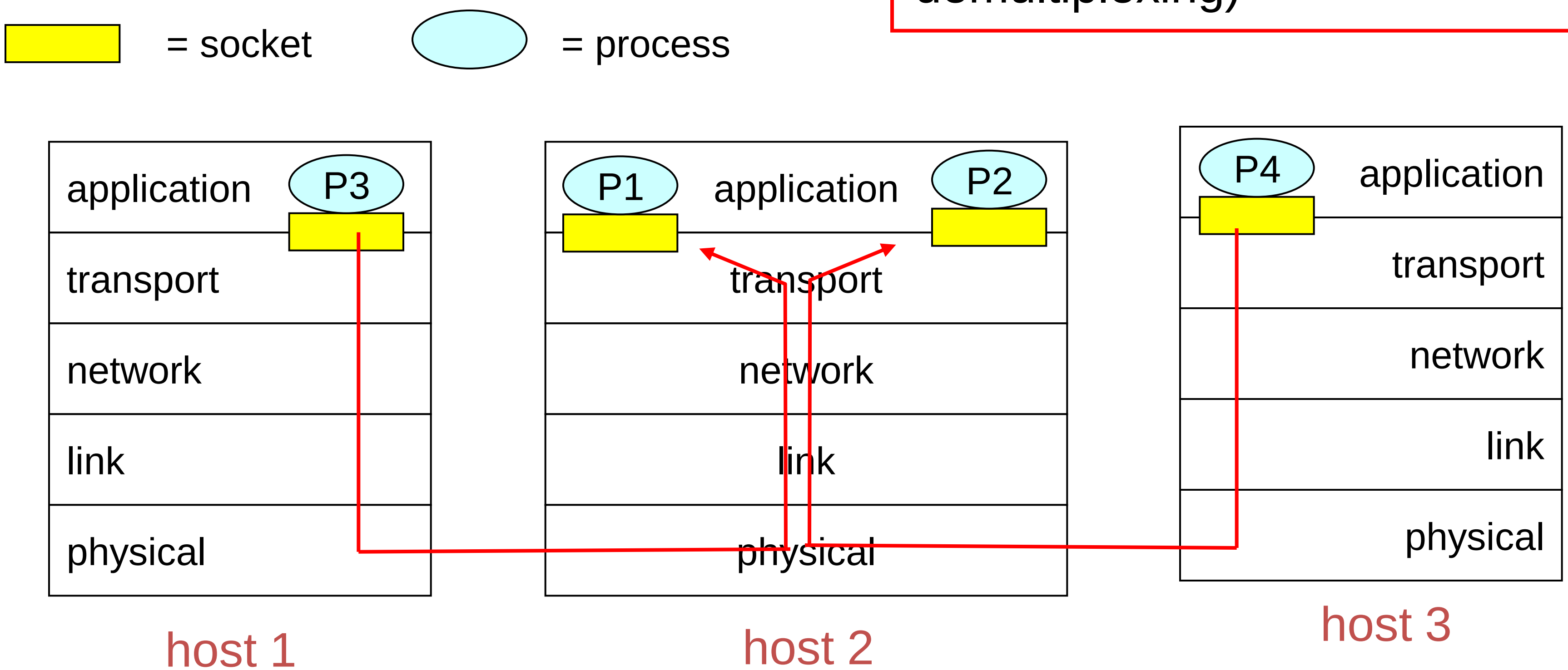
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

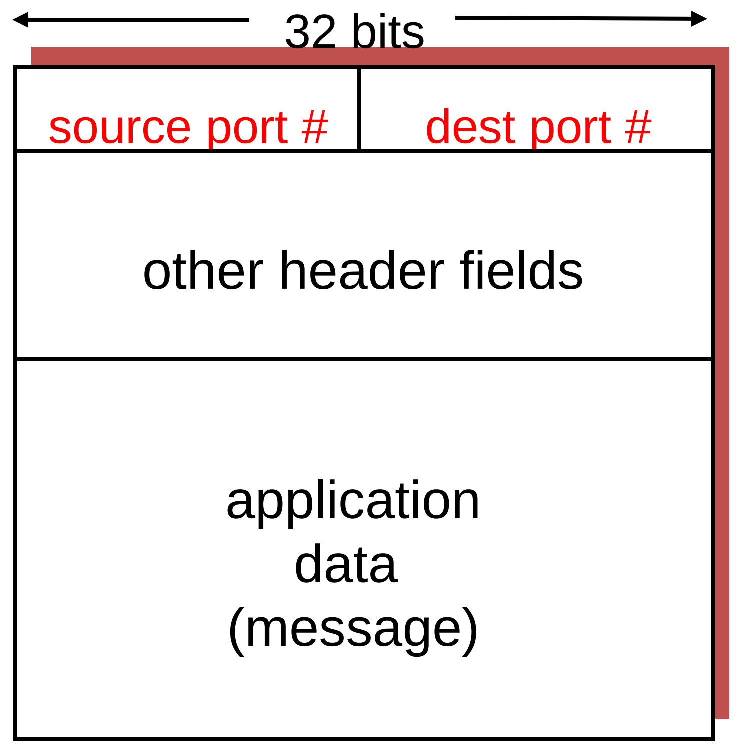


One HTTP process, one FTP process, one Telnet process

More than one socket, each socket has unique identifier

How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Analogous to car rentals at airports

Shuttles MUX passengers and take them
To rental office -- DeMUX to diff cars

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
    DatagramSocket(99111);
DatagramSocket mySocket2 = new
    DatagramSocket(99222);
```

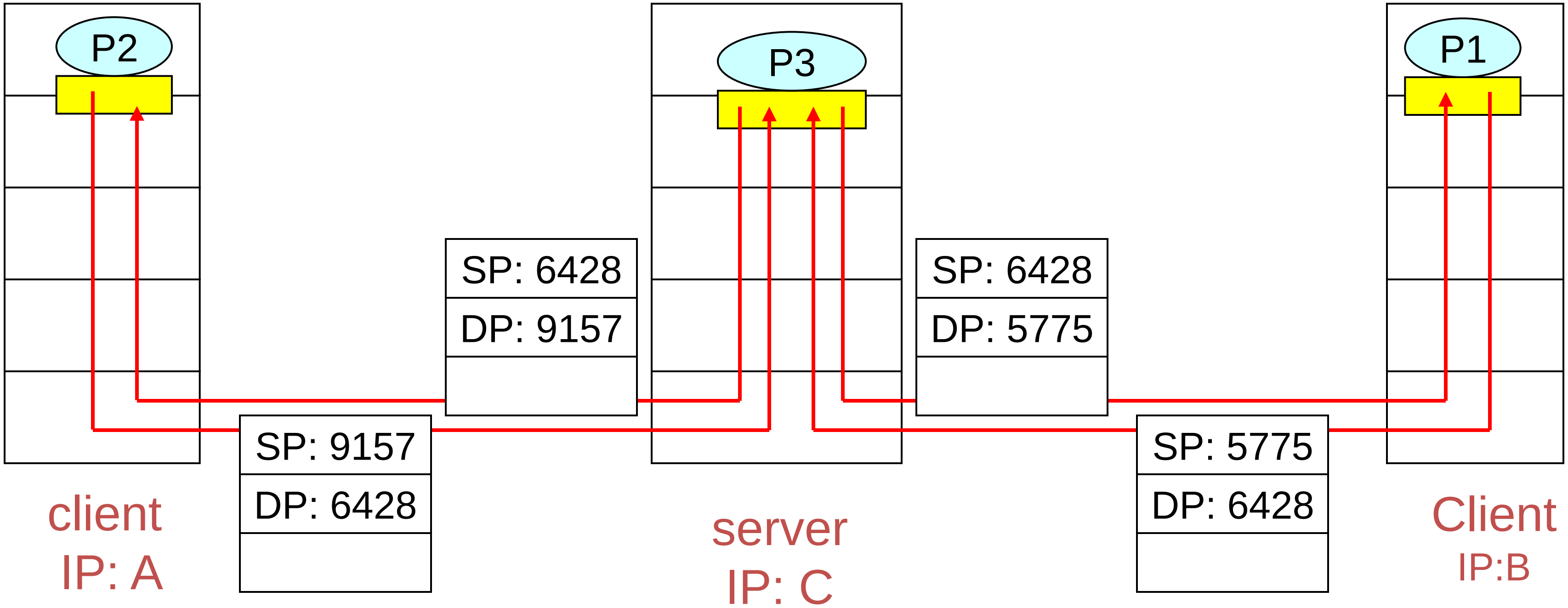
- UDP socket fully identified by two-tuple:

  (dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

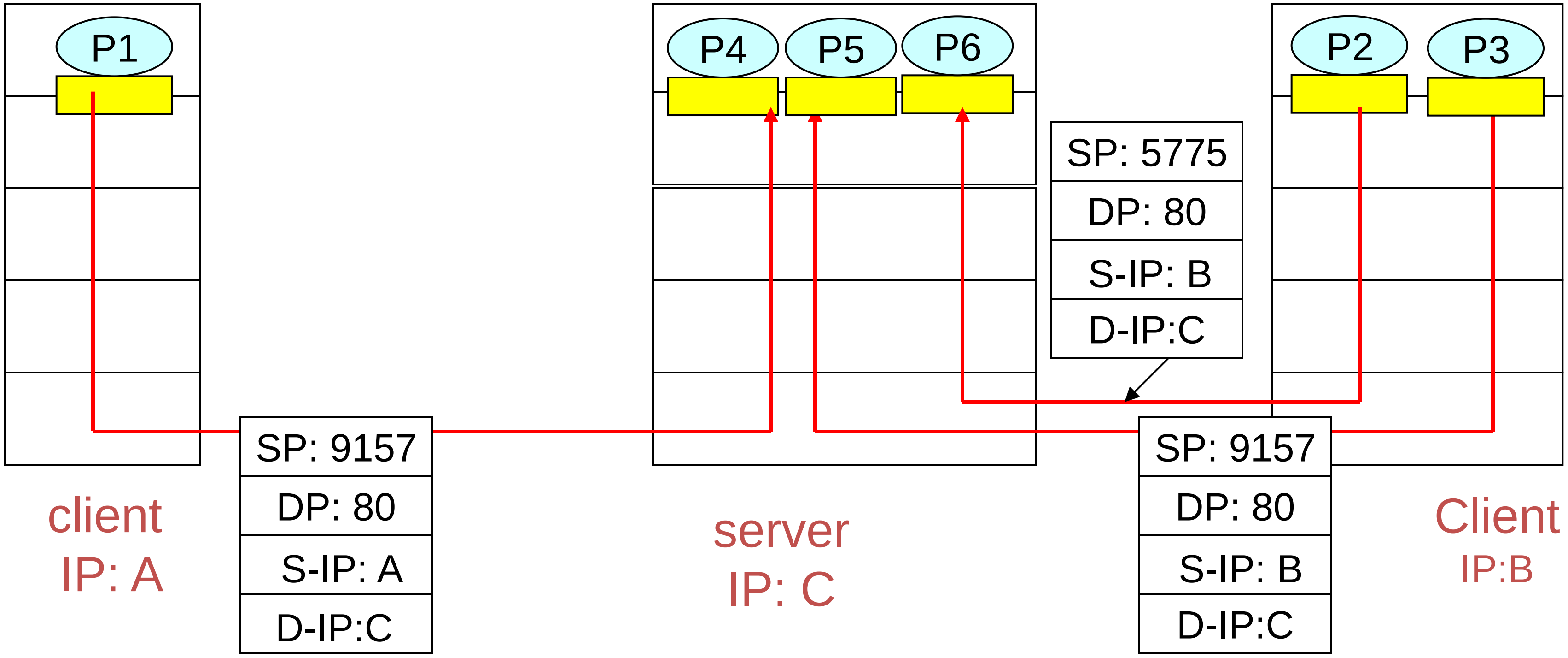


SP provides “return address”

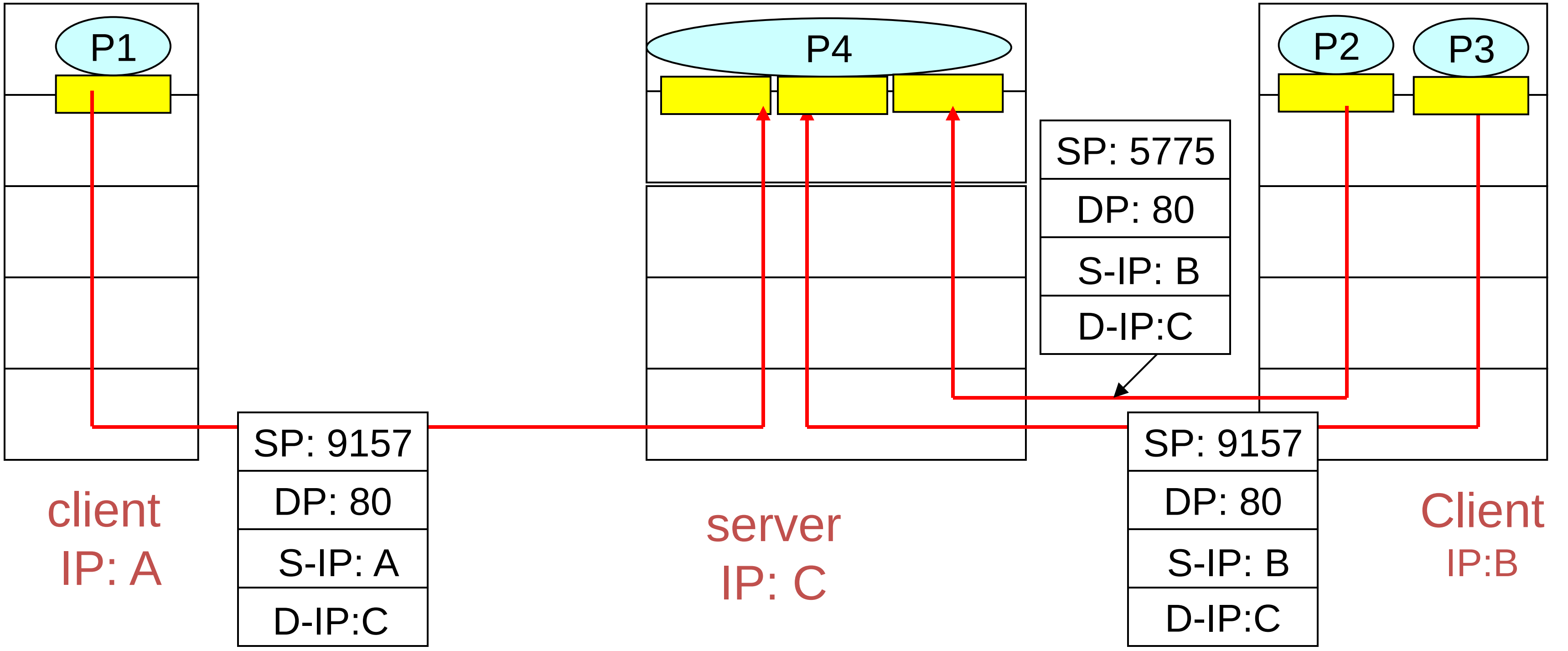
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux (cont)



Connection-oriented demux: Threaded Web Server



Chapter 3 Outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

Tentative Midterm Exam Structure

- Short Multiple Choice Questions = 15 points
- Chapter 1: (packet/circuit switching, Delay calculations etc.) 2*15 = 30 points
- Chapter 2: (HTTP, Email, DNS etc.) 20 + 10 = 30 points
- Chapter 3: (transport layer) 15 = 15 points
- General Concepts: 10 = 10 points

100 points

  - When: Tuesday (3/31) 4:30pm – 6:30pm
  - Where: In Class

Good Luck !